DATA STRUCTURES AND ALGORITHMS
Department of Electrical and Computer Engineering
Faculty of Science and Technology, University of Coimbra

WORKSHEET 3
PROCEDURAL PROGRAMMING IN C++

3.1 Objectives

This assignment aims at practicing procedural programming in C++, with a special emphasis on the aspects mentioned explicitly in the next section.

3.2 Introduction

Although many of the concepts involved in this assignment were already learnt by the students in the Computer Programming subject by using the C language, there are a few aspects that are differently, or slightly differently implemented in C++, namely:

- Comments within the source code;
- Input from keyboard and output to console;
- String manipulation;
- Input/Output from/to Files;
- Dynamic memory allocation.

There are also a few concepts that are entirely new in C++:

- Reference variable;
- Function returning a reference;
- Function overloading and default arguments;
- Namespace.

While in C, only `/*` and `*/` delimiters can be used to insert either single- or multi-line comments in the source code, the `//` delimiter can also be used in C++ to insert single-line comments.

The input from keyboard and output to console, as well as to files, appears differently in C++, compared with the C language, by using streams provided by `iostream` and `fstream` libraries. Also, the `iomanip` library provides a set of convenient stream manipulators that help a lot in formatting conveniently the text output, either on console or in text files.

The manipulation of C++ strings is also more intuitive and powerful than with C strings, by using the `string` data type provided by the library `string`.

Dynamic memory allocation is native in C++ language through the reserved words `new` and `delete`.

The C++ language allows defining function overloading, i.e. implementing multiple functions with the same name, provided that they differ in the list of arguments. The definition of default arguments is also possible in C++, providing a means to omit one or more arguments when calling a function with default arguments.

3.3 Passing arguments to a function by reference

Reference variables were introduced in C++ (they do not exist in C) and can be seen as alias to variables. They are important to pass arguments to functions by reference and to implement functions that return a reference.

A method (available in C++ but not available in C) to pass arguments to a function and allowing them to be changed within the function is to use argument passing by reference. When a reference is passed into a function, any change to that reference undergoes as if that change were made directly on the original variable.

The previous example seen in Worksheet 1, of a function that should swap the value of two integer variables, can be easily adapted to arguments passed by reference:

```
void swap(int &num1, int &num2) {
  int temp;
  temp = num2;
  num2 = num1;
  num1 = temp;
}
//----------------------------------------------------------
int main() {
  int x = 4, y = 2;

  printf("Before calling swap(), x=%d and y=%d\n", x, y);
  swap(x, y);
  printf(" After calling swap(), x=%d and y=%d\n", x, y);
}
```

3.4 Suggested Exercises

In the set of exercises listed below, you cannot use C functions whenever the programming feature in hands can be implemented differently in C++ (most often, in a more powerful way). For instance, you cannot use `scanf()` and `printf()` for input/output in this assignment; you have to use alternatively `cin` and `cout`.

Exercise 3.1 – `for` loop, `cout` – Medium
Write a function that, given an integer number as input argument, draws a diamond in the console whose side contains that number of asterisks. Write a small program (i.e. the `main()` function) to test your function. You can only use C++ methods for input/output (i.e. `cin` and `cout`). Check the usefulness of adding the line `using namespace std;` in your program. Example of the diamond obtained when the given number is 4:

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Exercise 3.2 – Input/Output, Output Formatting – Easy

Consider a function that multiplies a bi-dimensional matrix, containing real numbers represented in floating-point notation, by a real number (also a number represented in floating-point notation). Assume that the matrix has `COLUMNS =3` (a global constant) columns.

a) Choose an adequate prototype for the function and implement the function. Note that the function needs to receive the number of lines of the matrix as argument.

b) Write a program to test the function through the following example of a matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 1 & 0.5 \end{bmatrix}, \text{ with } \texttt{COLUMNS =3}.$$

The test program must ask the user to input a real number to be multiplied by the matrix and print the matrix before and after the function call. With this purpose, implement an auxiliary function to format adequately the output of the matrix on the console by using manipulators available in the `iomanip` library. Each matrix entry must be presented aligned to the right, in fixed-point notation with 3 digits after the decimal point. The matrix entries in the same column must appear properly aligned on the console. Assume that each matrix entry has up to 3 digits before the decimal point, i.e. it is limited to the interval]-1000.000; +1000.000[.

Exercise 3.3 – Functions, Text Files – Difficult **(appeared in the test held on 25/03/2015)**

Consider a structure (defined with the reserved word `struct`) that stores data about a customer of a dietetics center: `name` (C string, size 50), `female` (boolean flag that is `true` if the customer is a woman, `false` otherwise), `weight` (`float`), `height` (`float`) and `bmi` (`float` that stores the BMI – Body Mass Index).

a) Declare the `struct`.

b) Implement the function `computeBMI()` that receives as argument a structure passed by reference, computes the BMI and stores the result in the `bmi` field of the structure. Assume that the structure already contains valid values in `weight` and `height` fields.
Note: `bmi = weight / (height * height).`

c) Implement the function `updateBMI()` that receives as arguments a pointer to the first element of an array of structures and the size of the array, and updates the field `bmi` of every element in the array.
Note: this function must call the function `computeBMI()`.

d) Write the function `listCustomers()` that receives as arguments an array of customers, its size, a BMI threshold, and a filename. The function writes a text file that lists every customer in the array whose BMI is greater or equal to the threshold. The function returns the number of customers written to the file (zero if none); the function returns `-1` if some error occurs when opening the file. If the file already exists, data is added at the end of the file without destroying the pre-existent data. The data written in each line of the text file (name, weight and BMI of a customer) must be properly formatted by using manipulators of the `iomanip` library: name aligned to the left in column 1; weight and BMI aligned to the right in columns 60 and 70, using fixed-point notation with 2 digits after the decimal point.

e) Write a test program to test all the functions implemented in the previous points.

Exercise 3.4 – Functions, Binary Files, Dynamic Memory Allocation – Difficult

Consider again the `struct` datatype defined in exercise 3.3.

a) Implement a function that saves to a binary file all the data stored in an array of structures.

b) Implement a function that opens a binary file and retrieves the data of a customer stored in the binary file at the index `i`; assume that the index of the first customer is zero. If the function retrieves the data successfully, it returns `true`, otherwise it returns `false`.

c) Implement a function that adds data of a new customer at the end of a binary file.

d) Implement a function that reads all structures stored in a binary file and return them in an array. The function allocates dynamically memory for the array. The function returns a pointer to the first element of the created array and the size of the array.

e) Implement a `main()` function to test the functions implemented in points a), b), c), and d). The program should show a menu with the following options: 1-Input data of customers from keyboard to an array (implement an auxiliary function to input data of one customer from the keyboard and return a structure); 2-Save customers data to file; 3-Retrieve data of a customer from file; 4-Add new customer at end of file; 5-List all customers stored in file.

Exercise 3.5 – Strings – Medium

Implement a program that builds the list of the authors of a publication. The list of authors should be stored in memory as a C++ string. Program a loop to ask the user, in each iteration, the name of each author, with the format `"<first_name_author< <last_name_author>"`. The loop ends when the user inputs an empty string, i.e. an empty name. In the end of the loop execution, the result of the processing should be a string containing the list of authors with the format `"<first_name_author1> <last_name_author1>, ..., <first_name_authorN> <last_name_authorN>"`. Extend the program to abbreviate the first names in the string containing the list of authors, e.g. `"Winston Churchill, Franklin Roosevelt"` becomes `"W. Churchill, F. Roosevelt"`. Hint: The `method find_first_of()` allows searching for the first occurrence of a character at or after a given position in a string.

Exercise 3.6 – Function Overloading, Functions with Default Arguments – Easy

Consider a family of functions `<return_type> sum(…)` that compute the sum of a set of values; each function has a different argument list. Implement the functions indicated below and build a small program to call and test each one of them.

```
a) int sum(int a, int b, int c = 0);    // sum 2 or 3 integers (3rd argument optional)
b) int sum(int array[], int size);      // sum the elements of an array of integers
c) void sum(int array1[], int size1, const int array2[], int size2);  // sum 2 arrays
d) float sum(float a, float b, float c = 0.0);       // sum 2 or 3 floats
e) float sum(float array[], int size);   // sum the elements of an array of floats
f) void sum(float array1[], int size1, const float array2[], int size2);// sum 2 arrays
```

Exercise 3.7 – Reference Variable, Function – Medium
Explain why the following functions are not correct:

```
int &badIdea1() {
   int i;
   return i;
}
//---------------------------
int &badIdea2(int arg) {
   return arg;
}
```

If you do not know how to answer, create a small program that calls both functions, check the warnings provided by the compiler, and try to interpret them...

Exercise 3.8 – Arguments Passed by Reference to Functions – Easy

Write the function `void invert(int array[], int dim)` that swaps the position of the elements of the array of size `dim` (e.g. the first element becomes the last one, the second becomes the penultimate, etc.). Design its code so that it calls an auxiliary function `swapRef()` that swaps the value of the integer variables passed by reference as arguments to it.